1. APTs & CVEs (15 marks)

1.
   i) A legitimate use case for IFUNC is system call optimization. For example, gettimeofday() uses IFUNC to determine whether to use a kernel system call or the high-speed vDSO (Virtual Dynamically-linked Shared Object).

   ii) IFUNC was leveraged during the attack on xz-utils by hijacking the legitimate IFUNC resolver and calling the malicious code instead of choosing a CRC implementation. To start off, Jia Tan concealed the malicious binary payload in a test file that was included in the source tarball. A malicious script was then hidden in the **m4/build-to-host.m4** file, where, on the libzma library build (specifically, Linux x86-64 using FNU linker for Debian/RPM packages), the script extracted the malicious payload from the test files and injected it into the build process. The injected code replaced the IFUNC resolvers of **lzma_crc32** and **lzma_crc64** of **liblzma.so**, i.e., **crc32_resolve** and **crc64_resolve**, to add a call to a malicious **get_cpuid** method in place of the legitimate invocation of **cpuid**. The goal of the malicious resolver was to try to establish a Dynamic Linker Audit Hook, allowing the malicious code to intercept and modify the resolution of symbols in other libraries. The malicious code specifically targeted three functions: **RSA_public_decrypt**, **RSA_get0_key**, and **EVP_PKEY_set1_RSA**. If the backdoor finds a way to hook any of these functions, it will skip the rest, and it seems that the auditors were only able to get the **RSA_public_decrypt** hook to work, replacing calls to **RSA_public_decrypt** with **hook_main**. Once the hook is in place, an attacker connects via SSH and supplies a crafted public key. This triggers the backdoor, which verifies the attacker's hardcoded public key and then, if valid, bypasses the normal SSH authentication, enabling the attacker to execute arbitrary commands remotely.

2. Three shortcomings of open source projects for Linux are:
   Blind trust in test data. Developers often gloss over test files, allowing for malicious code to be introduced more easily through these channels. A solution is to limit what files and commands build scripts can access and execute, as well as mandate reproducible builds where every dependency is verified.
   Complex Linux build systems. The reliance on complex build systems like **Autotools** means malicious scripts hidden in files like **build-to-host.m4** are often missed because few developers understand the entire execution chain. To fix this, distributions could enforce the use of sandboxed build environments to make critical components more auditable.
   Lack of maintainer resources. Open source projects often rely on a few unpaid and overworked maintainers who may lack the time or resources to conduct deep security

reviews. Ideally, critical projects could use community funding and multi-person code reviews to reduce malicious attacks and reduce stress on the maintainers.

3. CVE-2024-29973
i) This is a command injection vulnerability affecting Zyxel Network Attached Storage devices. The logic handling of the "setCookie" parameter when processing an HTTP POST request allowed attackers to inject crafted requests with unsanitized input, which could execute shell commands.
ii) The base CWE for this is **CWE-78: Improper Neutralization of Special Elements used in an OS Command**

4. While the CVSS is highly reliable for measuring the technical severity of a vulnerability, it is often unreliable when used as the sole measure for organizational risk assessment. One issue is that CVSS intrinsically measures severity, not risk. It primarily quantifies the technical severity of a vulnerability based on things like exploitability and impact on CIA (Confidentiality, Integrity, and Availability). However, risk is a combination of exposure and asset criticality. This ties in to another issue: the lack of real-world threat context. The base CVSS score is static and does not reflect whether exploit code is publicly available or if the vulnerability is being actively exploited in the wild. While CVSS does offer optional Temporal Metrics to account for this change over time, most organizations rely on the static base score. Finally, there is the aspect of inconsistent and subjective interpretation. CVSS scores rely on the judgment of human assessors, and differences in interpreting metrics can lead to inconsistent scoring between different vendors or organizations. Furthermore, the lack of transparency from centralized databases like the NVD obscures how they arrive at their scores, which can result in different CVE numbering authorities assigning conflicting scores to the same vulnerability.

5. In my opinion, I do not think Mr. Super Secure's assessment is accurate. While he accurately highlights the operational costs of continuous updating and the prevalence of non-vulnerability attack vectors, his strategy introduces unnecessary and substantial security risks. Delaying general updates leaves systems vulnerable to every patchable flaw, significantly increasing the overall attack surface. While APTs might initially use spearphishing to gain access, unpatched software allows for privilege escalation or lateral movement once inside the network. The problem with "CVE-targeting" is that a vulnerability is only assigned a CVE *after* it is discovered and publicly disclosed. By waiting to update, you allow longer windows of time where an exploit is discovered but not published yet. These times are when an attack can strike. Another point is that software updates often contain patches for security bugs that aren't yet assigned a CVE or are a silent bug fix. A reactive approach misses these points. It's also important to factor in ecosystem compatibility: timely updates ensure software remains compatible with modern security tools. Finally, the financial cost, regulatory fines, and reputational damages from a major breach completely outweigh the inconvenience of regular updates.